Indonesian Journal of Electrical Engineering and Informatics (IJEEI)
Vol. 7, No. 4, Dec 2019, pp. 628~638
ISSN: 2089-3272, DOI: 10.11591/ijeei.v7i4.1146

Selecting Root Exploit Features Using Flying Animal-Inspired Decision

Ahmad Firdaus[1], Mohd Faizal Ab Razak[2], Wan Isni Sofiah Wan Din[3], Danakorn Nincarean[4], Shahreen Kasim[5], Tole Sutikno[6], Rahmat Budiarto[7]
[1,2,3,4]Faculty of Computer Systems & Software Engineering, University Malaysia Pahang, Kuantan, Pahang, Malaysia
[5]Faculty of Computer Science & Information Technology, Universiti Tun Hussein Onn Malaysia, Johor, Malaysia
[6]Department of Electrical and Computer Engineering, Universitas Ahmad Dahlan, Yogyakarta, Indonesia
[7]Department of Computer Science, Albaha University, Albaha, Saudi Arabia

Article Info

Article history:

Received Mar 26, 2019
Revised
Accepted Nov 13,2019

Keyword:

root exploit
Android
static analysis
machine learning
bee
bat
firefly

ABSTRACT

Malware is an application that executes malicious activities to a computer system, including mobile devices. Root exploit brings more damages among all types of malware because it is able to run in stealthy mode. It compromises the nucleus of the operating system known as kernel to bypass the Android security mechanisms. Once it attacks and resides in the kernel, it is able to install other possible types of malware to the Android devices. In order to detect root exploit, it is important to investigate its features to assist machine learning to predict it accurately. This study proposes flying animal-inspired (1) bat, 2) firefly, and 3) bee) methods to search automatically the exclusive features, then utilizes these flying animal-inspired decision features to improve the machine learning prediction. Furthermore, a boosting method (Adaboost) boosts the multilayer perceptron (MLP) potential to a stronger classification. The evaluation jotted the best result is from bee search, which recorded 91.48 percent in accuracy, 82.2 percent in true positive rate, and 0.1 percent false positive rate.

Corresponding Author:

Mohd Faizal Ab Razak
Faculty of Computer Systems and Software Engineering,
Universiti Malaysia Pahang,
26300 Gambang,
Kuantan, Pahang, Malaysia
Email: faizalrazak@ump.edu.my

1. INTRODUCTION

People utilize mobile devices in their daily activities to connect, online and communicate. This situation provides an opportunity for the attacker to develop root exploit to compromise victim's Android device for money or private purposes. Root exploit is an application software that takes over the kernel of the Android operating system to gain root privileges. When the attackers gain this privilege, they are able to provide false antivirus results, evade the Android security mechanisms, execute stealth activities without victim's acknowledgement, and install many types of malware to the devices [1]–[3]. In addition, the number of root exploits increasing from time to time because of the homebrew communities. These communities are the people that find multiple ways to break the Android kernel to obtain a customized version of Android. This leaves an opportunity for root exploit writers to wait for the homebrew community to discover new ways to gain control of the Android's kernel [4]. Consequently, in order to detect root exploit, security practitioners conducted the two types of malware analysis; 1) dynamic, and 2) static analysis.

Dynamic analysis investigates root exploit's behavior by executing the application and inspecting its movement [5][6]. The studies that practicing this type of analysis include the works in [7][8]. However, dynamic analysis has multiple drawbacks and one of it is the limited monitoring coverage. Because it monitors the application's characteristics within a certain time only. Therefore, root exploit's behaviors that running

exceeds the monitoring time of the dynamic analysis are excluded. This consequently leaves missing many root exploit's behaviors. In dissimilarity, static analysis diagnoses the root exploit by reverse engineering the application to retrieve its entire code. Security analysts practice this type of analysis without executing the malware. In addition, static analysis only needs few resources, which is low specifications of hardware, such as CPU, RAM, and storage. Moreover, static analysis process is fast which consume short amount of time than dynamic analysis [9]. During static analysis, the root exploit is unable to hide or modify its malicious process because it is unexecuted [10]. Nonetheless, in the interest to detect root exploit efficiently with machine learning, static analysis needs distinct features in minimal amount.

In machine learning intelligence prediction model, determining the optimal and the best features in fewer amounts increases the machine learning performance results. This performance increment is because fewer features eliminate unnecessary data and decreases the dataset's dimensionality. It also minimizes the nature of the predictive model, hence, reducing the machine learning processing time [11][12]. In the interest to have few relevant features, this paper adopts flying animal-inspired search approach, to intelligently investigates overall features and select the best features that focuses on detecting root exploit that undiscovered previously. This study utilizes categories of features that covers system command, directory path, and code-based. The first features, which is system command, it is a UNIX-based command in the operating system (OS). This study chooses this type of feature because it is permanent although the UNIX OS update its version regularly. The system command comprises of Android debug bridge (ADB) commands, executing processes and terminal commands. The following feature is the directory path, which consists of Linux kernel directories and system paths. The third feature is code-based features, which people use it for executing the commands, for instance, standard output (stdout), standard error (stderr), and standard input (stdin).

This study proposes three animal-inspired algorithms to search the relevant features in minimal amount. Then, in the experiments, this study uses Adaboost to boost and convert the multilayer perceptron into a strong learner for the machine learning classifiers to detect root exploit in Android mobile devices. In summary, this study has the following unique characteristics.

a) The use of 600 normal @ benign and 550 root exploit samples from the Malgenome dataset [13].
b) The utilization of three types of flying animal-inspired (bat, firefly, and bee) to automatically select the optimal @ best root exploit features that suits the multilayer perceptron machine learning classifier.
c) The utilization of multiple categories of features, which are system command, directory path and code-based features.
d) The use of Adaboost, a type of boost that converts the multilayer perceptron into a strong learner for efficient machine learning result.

The structure of this paper is as follows. Section 2 surveys the related works. Section 3 provides the methodology in the experiment. Section 4 presents the result derived from the experiment. Finally, section 5 delivers the conclusion and future works.

2. RELATED WORK

This section starts by introducing the root exploit and types of analysis to counter the type of malwares, and then followed by summary of previous researches related to static analysis and machine learning. The end of this section explains the flying animal-inspired method in selecting the optimal features and Adaboost that converts the MLP algorithm to efficiently predict the machine learning performance.

2.1 Root exploit

Unscrupulous authors or known as hackers construct malware to take over the Android operating system to gain private victim's information, stealing data, and eavesdropping communication. There are many types of malware, for instance, root exploit, spyware, botnet, Trojan, and worm. However, the most hazardous is the rootkit or known as root exploit [14], [15][16], [17]. It is because once the attackers take over the kernel with help of root exploit, all the OS layers are controlled by the attackers. Therefore, the infected OS will allow the attackers install multiple types of malware. In order to detect root exploit, researchers conduct malware analysis [18]–[20][21].

2.2 Malware analysis

The types of malware analysis are dynamic and static analysis. Dynamic analysis executes the malware and monitors it behaviors. The example of behaviors are user input and network traffic [22]–[30]. However, the limitations of dynamic analysis are, it needs high hardware specifications and consumes a lot of time to monitor the application one by one. Furthermore, it is also unable to detect the hidden activities during the monitor phase. Conversely, another type of malware analysis is static analysis, which examines application without monitor the behaviors [31][32].

Static analysis [33][6], [34] reverse engineers the application and inspecting its code. It covers unlimited coverage time because it does not execute the application [35]–[37]. The advantages of static analysis are; 1) covers the overall code, 2) inspect the overall structure of the application, 3) the analysis process is fast, and 4) able to detect unknown malware by combining with machine learning.

2.3 Static analysis and machine learning

Machine learning is a research that part of the artificial intelligence and provides the knowledge to the computers from the dataset, such as data observations and the environment interactions. From the data, it will allow the computers to predicts decisions and future judgements [38]. For example, [31] detected Android malware with Bayesian machine learning classification, with permissions as features. They utilized permissions as features that derived from Androidmanifest.xml as well as code-based.

In addition, a study by Shabtai et al. [39] used static analysis with features such as opcodes, string, methods and predicted by machine learning. Drebin et al. [40] practiced static analysis method and support vector machine (SVM). The authors utilized features such as permission, application programming interface (API) calls, xml files, network addresses to detect malware. However, the research excluded strings or keywords as features. Wei et al. introduced Droidexec [41] and Seo et al. introduced Droidanalyzer [42] also utilized static analysis in their studies. Droidexec [41] had adopted the graph constructor which uses opcode components, as features, while Droidanalyzer used API as well as keywords as features.

According to the authors' current knowledge, at the time of writing this paper, there are still lack of precious studies used flying animal-inspired (bat, firefly, and bee) to select the best features to detect malware whereas focus specifically on a root exploit. Therefore, this study utilizes the flying animal-inspired algorithm to select the best root exploit features.

2.4 Flying animal-inspired search

In machine learning prediction, it is important to selecting features to reduce model overfitting, to improve the performance of the machine learning prediction, and to shorten the model training time [43][44], [45]. The following sub-sections describe the flying animal inspired algorithms.

2.4.1 Bat search

Bat Search algorithm explores the best feature space based on the echolocation behavior of bats [46]. The bats use a type of sonar known as echolocation. It is fascinating as bats are able to search their prey and distinguish to different types of insects even in a complete darkness with help of echolocation. Other than searching their prey, they also used echolocation to avoid obstacles, and locate their roosting crevices in the dark. These bats emit a very loud sound pulse and listen for the echo that bounces back from the surrounding objects.

2.4.2 Firefly search

Firefly searches the best features based on the flashing pattern of tropical fireflies [47]. This algorithm searches the best features based on the brightness of the flash. For instance, any two fireflies that flash, the less bright firefly will fly towards the firefly that flashes much brighter. If the flash decreases, the instance between these two fireflies will increase as well. If there is no brighter than a particular firefly, it will move randomly.

2.4.3 Bee search

The bee algorithm searches the best features by following the bee strategy in finding the honey [48]. Bees strategy in finding the food source (honey) is by constructing two groups, called as scouts and foragers. Regularly, the quantity of the scouts is smaller than foragers. The bee in scout group will perform a signal known as waggle dance whenever they discover the food source (honey). The foragers will then fly towards to the food according to the signal from the scout. Some of the recruited foragers may also perform the waggle dance upon their return to the hive, mobilizing further foragers to exploit the food source.

Once the flying animal-inspired selected optimized features, this study further utilized it to classify and predict the root exploit malware with enhanced Multilayer perceptron with boosting method called Adaboost.

2.5 Adaboost

Boosting is a method to transform from powerless machine learning to a solid classifier. This study utilizes Adaboost to boost the Multilayer perceptron to enhance its execution in machine learning prediction.

Adaboost is referring to Adaptive Boosting. It is introduced by researchers [50] and this boosting algorithm is constructed according to the learning of the feeble calculation. It helps to deliver more exact and precise outcomes. Adaboost allocates each perception, x_i a fundamental weight respect, $w_i = \frac{1}{n}$, whereas n is the

number of insights. For each incorrect insight, w_i is prolonged while for each precise calculation, w_i is reduced. It practices the calculation repeatedly to train the weight of the fragile classifier till the point where the annotations are predicted. Section 3 explains the detail methodology that combines the flying animal-inspired with Adaboost MLP in detecting root exploit.

3. METHODOLOGY

Figure 1 depicts the proposed methodology that consists of four stages, which are data collection, reverse engineering, feature extraction, and machine learning classification. The initial step is to collect the dataset and followed by the reverse engineering. The steps followed by investigating and extracting the significant features and lastly the machine learning classification to detect the root exploit from the flying animal-inspired decision.

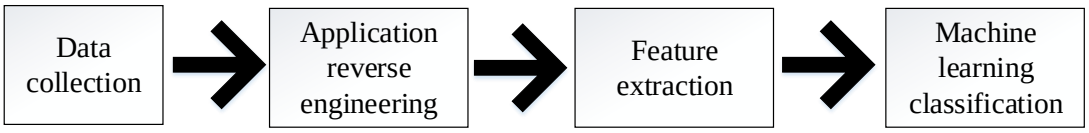

Figure 1. Methodology stages

3.1 Data Collection

Data collection phase needs two classes of applications comprises of normal @ benign and malware. As this step needs malware applications and extracted 1,260 samples of Malgenome dataset. This malware dataset consists of 49 families of malware [49] and many studies have utilized the dataset in their experiments [12, 14, 16]. Malgenome dataset consists of multiple of malware types (botnet, Trojan and root exploit). As this study emphasis on a root exploit malware only, therefore, the experiment extracted the only root exploit in this dataset, which comprises of 550 samples. Table 1 [13] lists the root exploit malware family and benign @ normal dataset.

Table 1. Information of root exploit and benign dataset

Root exploit dataset (Malgenome)	Frequency	Benign dataset (Google Play store)
Asroot	8	
BaseBridge	120	
DroidDream	16	
DroidDeluxe	1	
DroidCoupon	1	600
DroidKungfu 1	34	
DroidKungfu 2	30	
DroidKungfu 3	309	
DroidKungfu 4	20	
zHash	11	
Total (1150)	**550**	**600**

Other than malware, this study also needs a benign application for machine learning to distinguish between malware and benign. This study utilizes benign applications collected from the Google Play store [51]. The study collected 600 benign samples. Hence, the total of both malware and benign is 1,150 samples.

3.2 Reverse Engineering

The general process in static analysis is reverse engineering, which reverses the Android application to retrieve its native codes. The Android application uses .apk as their file extension. Figure 2 depicts the reverse engineering step that reverses the .apk to gain the Java codes by using Jadx [52]. Once we retrieve all the code, this step finds and grab the keywords by using "grep" command in Ubuntu terminal. Then, this step saved the result in .csv file. Once the grab process finishes, the following phase is to extract the root exploit features.

Figure 2**.** Reverse engineering process

3.3 Feature Extraction

The extracting features process includes investigating the suspicious strings in both malware and benign samples. As time is limited, this research only managed to find 31 features. Table 2 tabulates the features information which consists of three types of categories, the features in that categories and how many times the

features occur in both benign and malware. It lists the number of features in code-based is nine, directory path is 10 and system command is 12.

Table 2. List of features

No of features	Categories	Features	Benign	Malware
1	code-based	stdout	49	36
2		stdin	8	34
3		stderr	50	36
4		sePtyWindowSize	0	83
5		Forked	0	76
6		exec(su)	6	373
7		exec(sh)	2	9
8		exec()	191	427
9		createSubprocess	0	83
10	directory path	/system/xbin/su	38	361
11		/system/bin/su	16	363
12		/system/bin/sh	2	79
13		/system/bin/secbin	0	325
14		/system/bin/rm	0	17
15		/system/bin/profile	0	15
16		/system/bin/mount	0	31
17		/system/bin/chmod	12	377
18		/proc	4	16
19		/data/local/tmp/rootshell	0	15
20	system command	startservice n	0	359
21		ps	9	22
22		pm uninstall	0	1
23		pm install	0	61
24		mount o remount	2	92
25		kill	1	21
26		echo	18	87
27		cp rp	0	60
28		chown	0	79
29		chmod	23	187
30		cat	1	21
31		adb_enabled	3	360

The code-based category is features that comprise of general codes. For instance, setPtyWindowSize is a code to execute a process, stderr is a code to detect standard error and stdout is a code to output standard process in the operating system. Table 2 shows that one of the features, such as Forked, occur none in the benign application, instead occur 76 times in root exploit malware.

Meanwhile, the directory path category is referring to Android unique directory path whereas it based on Linux. It is because Android's kernel is based on Linux OS. As such, /system/xbin/su is the path that provides authorization to enter and receive access to the Linux kernel directories. The table tabulates one of the directory paths (/system/bin/chmod) that exist 12 times in benign, however, appeared 377 times in root exploit.

The next category is system command, which comprises of process, terminal and Android debug bridge (ADB). The ADB command is an application tool that enables the user to communicate with the Android emulator to connect to the Android devices [53]. One of the features is adb_enabled, which appeared three times in benign, but appeared 360 times in root exploit.

3.4 Feature Selection

In the interest to enhance the effectiveness of the machine learning in detecting root exploit, this study needs to discover the relevant features as minimal as possible. These relevant features will help to remove irrelevant and noisy data, hence, helps the performance of the multilayer perceptron's results [34], [54]–[56]. The implementation of the three flying animal-inspired algorithms (bat, firefly and bee) to find the best root exploit feature from the 31 features are shown in Table 3.

Table *3* shows that three flying animal-inspired algorithms (bat, firefly, and bee) choose different features. Among 31 features, Bat algorithm chooses 17, Firefly algorithm chooses 11 and Bee algorithm is the fewest than others, which is 7 features. After the feature selection was done, the next step is classification phase using MLP machine learning classifier to convert it to a strong learner with Adaboost.

Table 3. Flying animal-inspired features.

Features#	Flying animal-inspired algorithm: Bat	Firefly	Bee
1	adb_enabled	adb_enabled	chmod
2	chmod	chmod	exec(su)
3	chown	chown	setPtyWindowSize
4	createSubprocess	createSubprocess	startservice n
5	exec	exec(su)	/system/bin/chmod
6	exec(sh)	mount o remount	/system/bin/secbin
7	exec(su)	setPtyWindowSize	/system/bin/mount
8	mount o remount	startservice n	
9	pm install	/system/bin/chmod	
10	setPtyWindowSize	/system/bin/secbin	
11	startservice n	/system/bin/mount	
12	/system/bin/mount		
13	/system/bin/rm		
14	/system/bin/secbin		
15	/system/bin/su		
16	/system/xbin/su		
17	/system/bin/chmod		

3.5 The MLP Model

This step is to construct the machine learning predictive model. This is done by assembling the machine learning predictive model in the Weka (Waikato Environment for Knowledge Analysis) [57]. This phase performs this experiment in a machine that was furnished with Intel Core i7 as processor, Microsoft Windows 7 Professional and 16 GB of RAM.

4. RESULTS

This study uses cross validation as evaluation benchmark. In this process, 10-fold cross-validation takes place. The cross-validation method selects ten different parts of data randomly to two sets; (1) training; and (2) testing and these steps are repeated ten times. In each time, nine subsets were used for training set and one subset is for testing set. In particular, the testing set was omitted from the training set. In the attention to evaluate the flying animal-inspired features in detecting root exploit, Table 4 tabulates the evaluation in four types, 1) accuracy, 2) True Positive Rate (TPR), 3) False Positive Rate (FPR), and 4) ROC.

Table 4. List of evaluation measures.

	Evaluation measure	Descriptions	Equation
Greater value shows better classification	Accuracy	Correctly classify classes between malware or benign	$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$
	True positive rate (TPR)	Correctly classify classes as malware	$TPR = TP/(TP + FN)$
	ROC	The value closer to 1 indicates good classifier performance.	Tradeoff between two values, TPR and FPR
Lesser value shows better classification	False positive rate (FPR)	Incorrectly classify classes the class as malware, however it is actually benign	$FPR = FP/(FP + TN)$

4.1 Cross-validation

Table 5 tabulates the cross-validation results, while Figure 3 depicts these results in a graphical presentation for a clearer view. It covers the number of features, true positive rate (TPR), accuracy and receiver operating characteristic (ROC). The figure shows that the results are slightly similar even though the number of features is increasing. Nevertheless, this fact proves that the flying animal-inspired features (bat, firefly, and bat) are able to reach good prediction values, which exceed 91 percent in accuracy and 82 percent in TPR. In another point of view, this study observes another result, which is from the false positive rate (FPR) value.

Table 5. Classifier results in cross validation.

Evaluation measures	Adaboost with MLP			
	Bee	Firefly	Bat	Flying animal-inspired
	7	11	17	Number of features
Accuracy	91.48	91.39	91.04	Higher value indicates better performance
TPR	82.2	82	82.2	
ROC	91	91	91	
FPR	0.1	0.1	0.8	Lower value indicates better performance

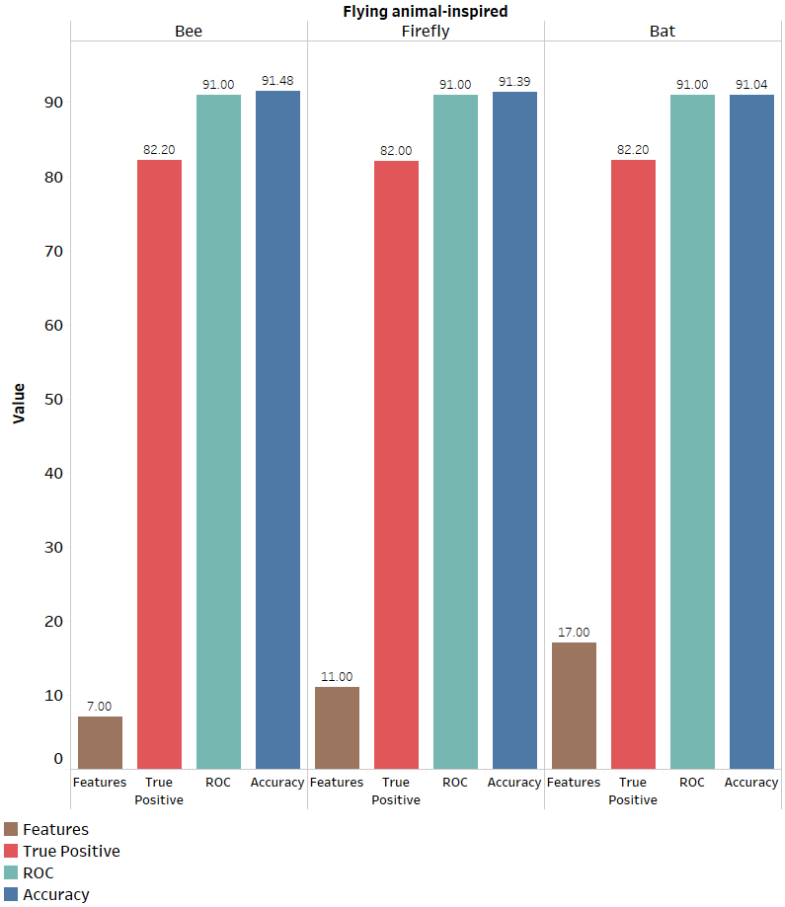

Figure 3. Accuracy, true positive and ROC results.

4.2 False positive rate

False positive rate (FPR indicates the value that incorrectly classify the class of the application as malware, however, it is actually normal @ benign. Therefore, the smaller the value, it is the best value. As it indicates the enhanced boost MLP did minor incorrect prediction. Figure 4 shows that, between bee, firefly and bat algorithms, bee algorithm did smaller mistakes than others, which jotted the best value (0.1 percent) with only seven features. While the bat algorithm marked the worst value which is 0.8 percent with 17 features.

Figure 4 shows the overall results of the FPR experiment. The figure proves that the fewer features utilization, the machine learning performance in prediction increases. As shown in the figure, the bat algorithm has chosen 17 features more than bee and firefly, hence did more mistakes and marked 0.8 percent. Meanwhile, the bee and bat algorithms decide to use only seven and eleven features, respectively and did lower mistakes than bat algorithm (0.1 percent).

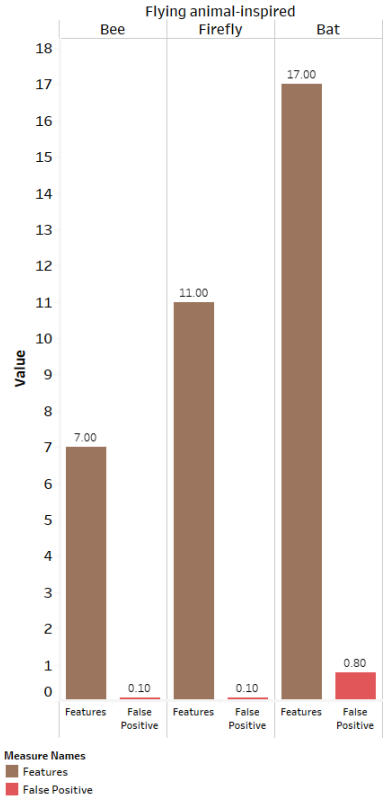

Figure 4. False positive results.

Figure 5 shows the plot between a number of features and false positive rate (FPR). The figure shows that the more features will lead to more incorrect prediction, whereas the trend line is rapidly increasing from 0.1 to 0.8 percent. This finding indicates that the additional features would lead to the higher incorrect percentage in FPR value. Accordingly, in the interest to achieve a good accuracy machine learning prediction, it is important to reduce the number of features and relevant as well.

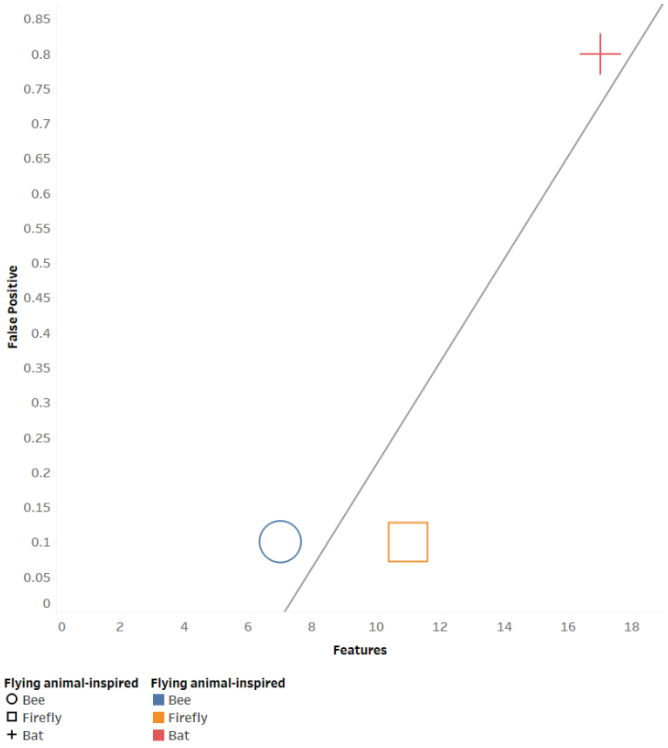

Figure 5. Combination plot between the number of features and false positive value.

5. CONCLUSION AND FUTURE WORKS

Root exploit is a malicious application that compromise the OS kernel to violate the root privileges of the OS. After it successfully attacks, it is capable to execute malicious activities without being notice, bypassing the authentication and install other types of malware. Accordingly, there is a need to predict the root exploit that undiscovered before. This study presented three flying animal-inspired to decide the best root exploit features consists of three categories, which are system command, directory path, and code-based. This study adopted the enhanced MLP with boost method called Adaboost to transform MLP into a strong machine learning classifier. From the evaluation, all the flying-animal inspired (bee, bat and firefly) algorithms have selected the optimal features and marked more than 91% accuracies in predicting unknown root exploit. However, in false positive rate results, with only seven features, bee jotted the lowest mistake among all algorithms, which is 0.1 % only. For future work, it is possible to add more types of features to increase the machine learning performance such as, extending the ADB features and other future studies by referring to works by [58][59][60].

ACKNOWLEDGEMENTS

This work was supported by Universiti Malaysia Pahang, under the Grant IBM Centre of Excellence (IBM2000) RDU180337.

REFERENCES

[1] Y. Ma and M. S. Sharbaf, "Investigation of Static and Dynamic Android Anti-virus Strategies," in *10th International Conference on Information Technology: New Generations (ITNG), Las Vegas, Nevada*, 2013, pp. 398–403.

[2] A. Schmidt *et al.*, "Smartphone Malware Evolution Revisited: Android Next Target?," in *IEEE Conference Publications, Montreal, Quebec, Canada*, 2009, pp. 1–7.

[3] J. Bickford, R. O'Hare, A. Baliga, V. Ganapathy, and Iftode Liviu, "Rootkits on smart phones: attacks, implications and opportunities," in *HotMobile '10 Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications, Annapolis, Maryland*, 2010, pp. 49–54.

[4] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM), Illinois, USA*, 2011, pp. 3–14.

[5] S. Khan, A. Gani, A. W. A. Wahab, and P. K. Singh, "Feature Selection of Denial-of-Service Attacks Using Entropy and Granular Computing," *Arabian Journal for Science and Engineering*, 2017.

[6] H. Tahaei, R. Salleh, M. F. A. Razak, K. Ko, and N. B. Anuar, "Cost Effective Network Flow Measurement for Software Defined Networks: A Distributed Controller Scenario," *IEEE Access*, pp. 1–17, 2018.

[7] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 343–357, Nov. 2014.

[8] F. Afifi, N. B. Anuar, S. Shamshirband, and K.-K. R. Choo, "DyHAP: Dynamic Hybrid ANFIS-PSO Approach for Predicting Mobile Malware," *Plos One*, vol. 11, no. 9, pp. 1–21, 2016.

[9] J. Lee, S. Lee, and L. Heejo, "Screening Smartphone Applications Using Malware Family Signatures," *Computers & Security*, vol. 52, pp. 234–249, 2015.

[10] A. Apvrille and T. Strazzere, "Reducing the window of opportunity for Android malware Gotta catch 'em all," *Journal in Computer Virology*, vol. 8, no. 1, pp. 61–71, Apr. 2012.

[11] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digital Investigation*, vol. 13, pp. 22–37, 2015.

[12] K. M. Alhendawi, "Predicting The Effectiveness Of Web Information Systems Using Neural Networks Modeling: Framework & Empirical Testing," *International Journal of Software Engineering and Computer Systems (IJSECS)*, vol. 4, no. 1, pp. 61–74, 2018.

[13] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *IEEE Symposium on Security and Privacy, San Francisco, CA*, 2012, no. 4, pp. 95–109.

[14] F. Ullah, M. Edwards, R. Ramdhany, R. Chitchyan, M. A. Babar, and A. Rashid, "Data Exfiltration: A Review of External Attack Vectors and Countermeasures," *Journal of Network and Computer Applications*, vol. 101, no. October 2017, pp. 18–54, 2018.

[15] Z. Tian, B. Wang, Z. Zhou, and H. Zhang, "The research on rootkit for information system classified protection," *2011 International Conference on Computer Science and Service System (CSSS)*, pp. 890–893, Jun. 2011.

[16] A. Firdaus and N. B. Anuar, "Root-exploit Malware Detection using Static Analysis and Machine Learning," in *Proceedings of the Fourth International Conference on Computer Science & Computational Mathematics (ICCSCM 2015), Langkawi, Malaysia*, 2015, pp. 177–183.

[17] A. Firdaus, N. B. Anuar, M. F. A. Razak, I. A. T. Hashem, S. Bachok, and A. K. Sangaiah, "Root Exploit Detection and Features Optimization: Mobile Device and Blockchain Based Medical Data Management," *Journal of Medical Systems*, vol. 42, no. 6, 2018.

[18] N. B. Anuar, M. Papadaki, S. Furnell, and N. Clarke, "An investigation and survey of response options for Intrusion Response Systems (IRSs)," in *Proceedings of the 9th Annual Information Security South Africa*

Conference, 2010, pp. 1–8.
[19] M. F. A. Razak, N. B. Anuar, R. Salleh, and A. Firdaus, "The rise of '"malware"': Bibliometric analysis of malware study," *Journal of Network and Computer Applications*, vol. 75, pp. 58–76, 2016.
[20] S. M. Zin, N. B. Anuar, M. L. M. Kiah, and A.-S. K. Pathan, "Routing protocol design for secure WSN: Review and open research issues," *Journal of Network and Computer Applications*, vol. 41, pp. 517–530, May 2014.
[21] H. A. S. Ahmed and M. F. Zolkipli, "Data security issues in cloud computing: review," *International Journal of Software Engineering and Computer Systems (IJSECS)*, vol. 2, no. February, pp. 58–65, 2016.
[22] A. Feizollah, N. B. Anuar, R. Salleh, F. Amalina, R. R. Ma'arof, and S. Shamshirband, "A Study Of Machine Learning Classifiers For Anomaly-Based Mobile Botnet Detection," *Malaysian Journal of Computer Science*, vol. 26, no. 4, pp. 251–265, 2013.
[23] N. Yaakob, I. Khalil, H. Kumarage, M. Atiquzzaman, and Z. Tari, "By-passing infected areas in wireless sensor networks using BPR," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1594–1606, 2015.
[24] A. Shabtai, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Computers & Security*, vol. 43, pp. 1–18, 2014.
[25] Y. Lin, Y. Lai, C. Chen, and H. Tsai, "Identifying android malicious repackaged applications by thread-grained system call sequences," *Computers & Security*, vol. 39, pp. 340–350, 2013.
[26] A. Feizollah, S. Shamshirband, N. B. Anuar, R. Salleh, and M. L. M. Kiah, "Anomaly Detection Using Cooperative Fuzzy Logic Controller," in *16th FIRA RoboWorld Congress (FIRA), Kuala Lumpur, Malaysia*, 2013, pp. 220–231.
[27] L. Xie, X. Zhang, J.-P. Seifert, and S. Zhu, "pBMDS : A Behavior-based Malware Detection System for Cellphone Devices," in *3rd ACM Conference on Wireless Network Security Location: Stevens Institute Technology, Hoboken, NJ*, 2010, pp. 37–48.
[28] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-Based Malware Detection System for Android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, Chicago, Illinois, USA*, 2011, pp. 15–26.
[29] A. Feizollah, N. B. Anuar, R. Salleh, and F. Amalina, "Comparative Study of K-means and Mini Batch K-means Clustering Algorithms in Android Malware Detection Using Network Traffic Analysis," in *International Symposium on Biometrics and Security Technologies (ISBAST)*, 2014, no. February.
[30] A. A. Allahham and M. A. Rahman, "A Smart Monitoring System For Campus Using Zigbee Wireless Sensor Networks," *International Journal of Software Engineering and Computer Systems (IJSECS)*, vol. 4, no. 1, pp. 1–14, 2018.
[31] S. Y. Yerima, S. Sezer, and G. McWilliams, "Analysis of Bayesian classification-based approaches for Android malware detection," *IET Information Security*, vol. 8, no. 1, pp. 25–36, 2014.
[32] A. Firdaus, N. B. Anuar, M. F. A. Razak, and A. K. Sangaiah, "Bio-inspired computational paradigm for feature investigation and malware detection: interactive analytics," *Multimedia Tools and Applications*, 2017.
[33] B. Chess and G. McGraw, "Static analysis for security," *IEEE Security & Privacy Magazine*, vol. 2, no. 6, pp. 76–79, 2004.
[34] M. F. A. Razak, N. B. Anuar, F. Othman, A. Firdaus, F. Afifi, and R. Salleh, "Bio-inspired for Features Optimization and Malware Detection," *Arabian Journal for Science and Engineering*, 2017.
[35] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras, and W. Lee, "Eureka: A framework for enabling static malware analysis," in *Lecture Notes in Computer Science*, vol. 5283, 2008, pp. 481–500.
[36] T.-K. Chang and G.-H. Hwang, "The design and implementation of an application program interface for securing XML documents," *Journal of Systems and Software*, vol. 80, no. 8, pp. 1362–1374, 2007.
[37] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android," in *Security and Privacy in Communication Networks*, 2013, pp. 86–103.
[38] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," *Informatica*, vol. 31, pp. 249–268, 2007.
[39] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated Static Code Analysis for Classifying Android Applications Using Machine Learning," in *Ninth International Conference on Computational Intelligence and Security, Nanning, Guangxi Zhuang Autonomous Region China*, 2010, pp. 329–333.
[40] D. Arp, M. Spreitzenbarth, H. Malte, H. Gascon, and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," in *21th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA*, 2014, pp. 1–15.
[41] T. Wei, H. Lee, H.-R. Tyan, H. M. Liao, A. B. Jeng, and J. Wang, "DroidExec: Root Exploit Malware Recognition Against Wide Variability via Folding Redundant," in *17th International Conference Advanced Communication Technology (ICACT), PyeongChang, Korea*, 2015, pp. 161–169.
[42] S.-H. Seo, A. Gupta, A. Mohamed Sallam, E. Bertino, and K. Yim, "Detecting mobile malware threats to homeland security through static analysis," *Journal of Network and Computer Applications*, vol. 38, pp. 43–53, 2014.
[43] S. Fong, R. P. Biuk-Aghai, and R. C. Millham, "Swarm search methods in weka for data mining," in *Proceedings of 10th International Conference on Machine Learning and Computing (ICMLC)*, 2018, pp. 122–127.
[44] Z. Mustaffa and Y. Yusof, "A Hybridization of Enhanced Artificial Bee Colony-Least Squares Support Vector Machines for Price Forecasting," *Journal of Computer Science*, vol. 8, no. 10, pp. 1680–1690, 2012.
[45] Z. Mustaffa, M. H. Sulaiman, and M. N. M. Kahar, "LS-SVM hyper-parameters optimization based on GWO algorithm for time series forecasting," in *4th International Conference on Software Engineering and Computer Systems, ICSECS 2015: Virtuous Software Solutions for Big Data, Kuantan, Pahang*, 2015, pp. 183–188.

[46] X.-S. Yang, "A New Metaheuristic Bat-Inspired Algorithm," in *Nature Inspired Cooperative Strategies for Optimization (NICSO)*, vol. 284, 2010, pp. 65–74.
[47] X. S. Yang and X. He, "Firefly algorithm: recent advances and applications," *International Journal of Swarm Intelligence*, vol. 1, no. 1, p. 36, 2013.
[48] D. T. Pham, M. Castellani, and H. A. Le-Thi, "The Bees Algorithm: Modelling Nature To Solve Complex Optimisation Problems," in *Proceedings of the 11th International Conference on Manufacturing Research (ICMR2013)*, 2013, pp. 481–488.
[49] Y. Zhou and X. Jiang, "Android Malware Genome Project," 2012. .
[50] Y. Zhongyang, Z. Xin, B. Mao, and L. Xie, "DroidAlarm: An All-sided Static Analysis Tool for Android Privilege-escalation Malware," in *Proceedings of Computer and Communications Security (CCS), Hangzhou, China*, 2013, pp. 353–358.
[51] Google, "Google Play Store," 2014. .
[52] Skylot, "Jadx," 2015. .
[53] Android Developer, "Android Debug Bridge (ADB)," 2017. .
[54] R. Jensen and Q. Shen, *Computational Intelligence and Feature Selection: Rough and Fuzzy Approaches*. Wiley-IEEE Press, 2008.
[55] K. S. Adewole, N. B. Anuar, A. Kamsin, K. D. Varathan, and S. A. Razak, "Malicious accounts: Dark of the social networks," *Journal of Network and Computer Applications*, vol. 79, pp. 41–67, 2017.
[56] A. Firdaus, N. B. Anuar, A. Karim, and M. F. A. Razak, "Discovering optimal features using static analysis and genetic search based method for android malware detection," *Frontiers of Information Technology & Electronic Engineering*, pp. 1–27, 2017.
[57] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *ACM SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.
[58] M. K. Khaleel, M. A. Ismail, U. Yunan, and S. Kasim, "Review on Intrusion Detection System Based on The Goal of The Detection System," *International Journal of Integrated Engineering*, vol. 10, no. 6, 2018.
[59] M. A. Ismail, V. Mezhuyev, K. Moorthy, S. Kasim, and A. O. Ibrahim, "Optimisation of biochemical systems production using hybrid of newton method, differential evolution algorithm and cooperative coevolution algorithm," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 8, no. 1, 2017.
[60] M. A. Ismail, V. Mezhuyev, S. Deris, M. S. Mohamad, S. Kasim, and R. R. Saedudin, "Multi-objective optimization of biochemical system production using an improve Newton Competitive differential evolution method," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 7, no. 4–2 Special Issue, 2017.